High-level development and debugging of FPGA-based network programs

nik.sultana@cl.cam.ac.uk

Intel Lynnfield
(c) Intel
Field-programmable gate arrays

Xilinx XC2064 FPGA
64 CLB x (2 3-LUTs + FF)
(c) Xilinx

http://ca.olin.edu/2005/fpga_dsp/fpga.html
• FPGAs becoming more **powerful** and **prevalent**. Most recently in datacentres.
  o e.g., Azure, Baidu, Amazon.
• FPGAs becoming more **powerful** and **prevalent**.
  Most recently in datacentres.

• But still **difficult** to program and debug!
  To both **hardware** and **non-hardware** people.
  Both **technical** and **non-technical** difficulties.

○ Hardware programming unlike software programming.
○ Generating a bitstream is a lengthy process.

○ Differing interpretations for standard HDLs.
○ Quality of the tools is less polished than for software.
○ Strong vendor bias, closed formats, hold things back.
• FPGAs becoming more **powerful** and **prevalent**.
  Most recently in datacentres.
• But still **difficult** to program and debug!
  To both **hardware** and **non-hardware** people.
  Both **technical** and **non-technical** difficulties.
• 30+ years of research into using **high-level languages**
  for circuit description. 20+ years commercial tooling
  (Synopsys Behavioural Compiler in 1994)
FPGAs becoming more powerful and prevalent.
Most recently in datacentres.
But still difficult to program and debug!
To both hardware and non-hardware people.
Both technical and non-technical difficulties.
30+ years of research into using high-level languages for circuit description. 20+ years commercial tooling (Synopsys Behavioural Compiler in 1994)
Experience has been mixed — you can’t be everything to everybody! But lots of progress.
High-level development and debugging of FPGA-based network programs
High-level development and debugging of FPGA-based network programs
Gates (+ Interconnections)
Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)
General Purpose Language (High-Level Synthesis) (e.g., C)

Hardware Description Language (e.g., Verilog)

Gates (+ Interconnections)
Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)
Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)
Hardware Description
Language

• Lots of **flexibility**.

• How you **think** code will behave vs how it’s **translated** vs how it **executes/implements**.

  • “reg” ~ register

  • “inference”

  • “technology mapping”
struct node {
    unsigned int prev_node : IDX_WIDTH;
    unsigned int next_node : IDX_WIDTH;
    unsigned int data      : DATA_WIDTH;
}

struct node memory [MAX_DEPTH_IDX+1];

`define DATA_F_LSB 0
`define DATA_F_MSB (`DATA_F_LSB + DATA_WIDTH - 1)
`define DATA_F_WORD `DATA_F_MSB:`DATA_F_LSB

`define NEXT_NODE_F_LSB (`DATA_F_MSB + 1)
`define NEXT_NODE_F_MSB (`NEXT_NODE_F_LSB + IDX_WIDTH - 1)
`define NEXT_NODE_F_WORD `NEXT_NODE_F_MSB:`NEXT_NODE_F_LSB

`define PREV_NODE_F_LSB (`NEXT_NODE_F_MSB + 1)
`define PREV_NODE_F_MSB (`PREV_NODE_F_LSB + IDX_WIDTH - 1)
`define PREV_NODE_F_WORD `PREV_NODE_F_MSB:`PREV_NODE_F_LSB

reg [`PREV_NODE_F_MSB:`DATA_F_LSB] memory [MAX_DEPTH_IDX:0];
Domain Specific Language
(e.g., PP, PX, P4)

General Purpose Language
(High-Level Synthesis)
(e.g., C)

Hardware Description Language
(e.g., Verilog)

Gates (+ Interconnections)
Domain-Specific Language

- Much **less flexibility**. Must stay within “domain”.

- Can achieve **good performance** and more **development support** (e.g., richer types), and **shorter cycles** of development.

- **Tuning** can be tricky — e.g., breakout to HDL.
Domain Specific Language  
(e.g., PP, PX, P4)

General Purpose Language  
(High-Level Synthesis)  
(e.g., C)

Hardware Description Language  
(e.g., Verilog)

Gates (+ Interconnections)
High-Level Synthesis

• Use “familiar” language.

• Usually not the full language. e.g., dynamic allocation only partly supported.

• Often involves library support and language extensions.

• Tuning can be tricky — e.g., breakout to HDL.
A: \( x = \) (some expression)
B: \( y = \) (some expression)

...
A: \( x = \text{(some expression)} \)
B: \( y = \text{(some expression)} \)
A: $x = \text{(some expression)}$
C: $f(x)$
B: $y = \text{(some expression)}$
...
A: $x = \text{(some expression)}$
C: $f(x)$
B: $y = \text{(some expression)}$
...
A: $x = (\text{some expression})$
B: $y = (\text{v.cplx expression})$

...
A: $x = (\text{some expression})$
B: $y = (\text{some expression})$

...
A: $x = \text{(some expression)}$
B: $y = \text{(some expression)}$

...
A: \( x = \text{(some expression)} \)
B: \( y = \text{(some expression)} \)

...
“The user can control how aggressively Stratus HLS packs these operations into each clock period. Creating designs with Stratus HLS can save months of backend effort by preventing timing closure problems.”
Goal

High-level development and debugging of FPGA-based network programs

...using HLS
It won’t work…
It won’t work…

• **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.
It won’t work…

• **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

• **HLS tools are expensive and closed-source.**
  Various academic tools exist.
It won’t work…

• **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

• **HLS tools are expensive and closed-source.**
  Various academic tools exist.

• **Not sufficiently expressive.**
  Can be fixed. (With ingenuity.)
It won’t work…

• **Performance will suck!**
  Use HDL modules from HLL for resource-related IP.

• **HLS tools are expensive and closed-source.**
  Various academic tools exist.

• **Not sufficiently expressive.**
  Can be fixed. (With ingenuity.)

• **How to run this in software?**
  Create emulation environment + shadow library.
Concerns

• Providing benefits for **hardware people**: improved time-to-market, prototyping, development support, debugging, can breakout to HDL.
Concerns

• Providing benefits for **hardware people**: improved time-to-market, prototyping, development support, debugging, can breakout to HDL.

• Providing benefits for **non-hardware people**: through less steep learning curve, software-like development mindset.
Concerns

• Providing benefits for **hardware people**: improved time-to-market, prototyping, development support, debugging, can breakout to HDL.

• Providing benefits for **non-hardware people**: through less steep learning curve, software-like development mindset.

• Comparable or better **performance** (latency + throughput) or resource **utilisation** to hand-written HDL.
Our system: **Emu**

(High-level) **Software** description of network program

**Hardware** description of network program
(1) HLS

(High-level) Software description of network program

C#

Hardware description of network program

Verilog

http://www.cl.cam.ac.uk/~djg11/kiwi/
public class Data {
    public bool matched = false;
    public ulong result = 0;
}

public class LRU
{
    public static Data Lookup(ulong key_in)
    {
        Data res = new Data();
        ulong idx = HashCAM.Read(key_in);
        if (HashCAM.matched) {
            res.matched = HashCAM.matched;
            res.result = NaughtyQ.Read(idx);
            NaughtyQ.BackOfQ(idx);
        }
        return res;
    }

    public static void Cache(ulong key_in, ulong value_in)
    {
        ulong idx = NaughtyQ.Enlist(value_in);
        HashCAM.Write(key_in, idx);
    }
}
(2) Library support

(High-level)
Software
description
of network
program

C#
+ libraries

Kiwi

Hardware
description
of network
program

Verilog
+ libraries
(3) Host environment

(High-level) Software description of network program

C# + libraries

Hardware description of network program

Verilog + libraries

http://github.com/niksu/Pax
private ForwardingDecision OutsideToInside(TEncapsulation packet)
{
    // Retrieve the mapping. If a mapping doesn't exist, then it means that we're not aware of a session to which the packet belongs: so drop the packet.
    var key = new ConnectionKey(packet.GetSourceNode(), packet.GetDestinationNode());
    NatConnection<TPacket,TNode> connection;
    if (NAT_MapToInside.TryGetValue(key, out connection))
    {
        var destination = connection.InsideNode;

        // Update any connection state, including resetting the inactivity timer
        connection.ReceivedPacket(packet, packetFromInside: false);

        // Rewrite the packet destination
        packet.SetDestination(destination);

        // Update checksums
        packet.UpdateChecksums();

        // Forward on the mapped network port
        return new ForwardingDecision.SinglePortForward(destination.InterfaceNumber);
    }
    else
    {
        return Drop;
    }
}
(3) Hardware envir.

(High-level) Software description of network program

C#
  + libraries

Hardware description of network program

Verilog
  + libraries

http://netfpga.org/
Lifting & shadowing

(High-level) Software description of network program

C# + libraries

Hardware description of network program

Verilog + libraries
protected static bool enable;
protected static bool ready;
protected static bool crashed;

protected static byte command;

protected static ulong idx_out;
protected static ulong data_out;

protected static ulong idx_in;
protected static ulong data_in;

// Nonvolatile copies of outputs.
public static ulong idx_out_nv;
public static ulong data_out_nv;

public static ulong Enlist(ulong data_in)
{
    while (ready) { Kiwi.Pause(); }
    command = (byte)op_code.Enlist;
    NaughtyQ.data_in = data_in;
    enable = true;
    Kiwi.Pause();
    while (!ready) { Kiwi.Pause(); }
    Kiwi.Pause();
    idx_out_nv = idx_out;
    data_out_nv = data_out;
    enable = false;
    Kiwi.Pause();
    return idx_out_nv;
}
protected static bool enable;
protected static bool ready;
protected static bool crashed;

protected static byte command;

protected static ulong idx_out;
protected static ulong data_out;

protected static ulong idx_in;
protected static ulong data_in;

// Nonvolatile copies of outputs.
public static ulong idx_out_nv;
public static ulong data_out_nv;
public static ulong Enlist(ulong data_in)
{
    while (ready) { Kiwi.Pause(); }
    command = (byte)op_code.Enlist;
    NaughtQ.data_in = data_in;
    enable = true;
    Kiwi.Pause();
    while (!ready) { Kiwi.Pause(); }
    Kiwi.Pause();
    idx_out_nv = idx_out;
    data_out_nv = data_out;
    enable = false;
    Kiwi.Pause();
    return idx_out_nv;
}
public static ulong Enlist(ulong data_in)
{
    while (ready) { Kiwi.Pause(); }
    command = (byte)op_code.Enlist;
    NaughtyQ.data_in = data_in;
    enable = true;
    Kiwi.Pause();
    while (!ready) { Kiwi.Pause(); }
    Kiwi.Pause();
    idx_out_nv = idx_out;
    data_out_nv = data_out;
    enable = false;
    Kiwi.Pause();
    return idx_out_nv;
}
public static ulong Enlist(ulong data_in)
{
    while (ready) { Kiwi.Pause(); }
    command = (byte)op_code.Enlist;
    NaughtyQ.data_in = data_in;
    enable = true;
    Kiwi.Pause();
    while (!ready) { Kiwi.Pause(); }
    Kiwi.Pause();
    idx_out_nv = idx_out;
    data_out_nv = data_out;
    enable = false;
    Kiwi.Pause();
    return idx_out_nv;
}
public static ulong Enlist(ulong data_in)
{
    while (ready) { Kiwi.Pause(); }
    command = (byte)op_code.Enlist;
    NaughtyQ.data_in = data_in;
    enable = true;
    Kiwi.Pause();
    while (!ready) { Kiwi.Pause(); }
    Kiwi.Pause();
    idx_out_nv = idx_out;
    data_out_nv = data_out;
    enable = false;
    Kiwi.Pause();
    return idx_out_nv;
}
Signals

<table>
<thead>
<tr>
<th>Time</th>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>0</td>
</tr>
<tr>
<td>command[3:0]</td>
<td>1</td>
</tr>
<tr>
<td>crashed</td>
<td>0</td>
</tr>
<tr>
<td>data_in[7:0]</td>
<td>02</td>
</tr>
<tr>
<td>data_out[7:0]</td>
<td>00</td>
</tr>
<tr>
<td>enable</td>
<td>1</td>
</tr>
<tr>
<td>idx</td>
<td>00000010</td>
</tr>
<tr>
<td>idx_in[3:0]</td>
<td>x</td>
</tr>
<tr>
<td>idx_out[3:0]</td>
<td>x</td>
</tr>
<tr>
<td>ready</td>
<td>0</td>
</tr>
<tr>
<td>reset</td>
<td>0</td>
</tr>
</tbody>
</table>

Waves

- clk: 0
- command[3:0]: 1
- crashed: 0
- data_in[7:0]: 02
- data_out[7:0]: 00
- enable: 1
- idx: 00000010
- idx_in[3:0]: x
- idx_out[3:0]: x
- ready: 0
- reset: 0
Signals

<table>
<thead>
<tr>
<th>Time</th>
<th>Waves</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk = 0</td>
<td></td>
</tr>
<tr>
<td>command[3:0] = 1</td>
<td></td>
</tr>
<tr>
<td>crashed = 0</td>
<td></td>
</tr>
<tr>
<td>data_in[7:0] = 02</td>
<td></td>
</tr>
<tr>
<td>data_out[7:0] = 00</td>
<td></td>
</tr>
<tr>
<td>enable = 1</td>
<td></td>
</tr>
<tr>
<td>idx = 00000010</td>
<td></td>
</tr>
<tr>
<td>idx_in[3:0] = x</td>
<td></td>
</tr>
<tr>
<td>idx_out[3:0] = 0</td>
<td></td>
</tr>
<tr>
<td>ready = 0</td>
<td></td>
</tr>
<tr>
<td>reset = 0</td>
<td></td>
</tr>
</tbody>
</table>
Input Arbiter
Packet processing logic
Output Queues

Dataplane

Physical interface, queues and other components
The design is tested using a **testbench** that simulates the hardware being fed a set of packets, and checking the packets that result.

The design is ultimately processed by FPGA vendor-specific tools to generate an image that programs the FPGA. Packets received on physical network ports are processed using our logic.
Packet processor modelled in C# 

Design is tested using a testbench that simulates the hardware being fed a set of packets, and checking the packets that result.

.NET bytecode can be executed in .NET VM on various OSs, and debugged using existing tools.

Network-scale testing can be done cheaply and easily within a single machine.

Layers of abstraction between the .NET VM and the OS-provided virtual or physical network interfaces.

Virtualisation of interfaces enables us to use the packet processor inside a network simulator.

HDL-based development and integration stages

The design is ultimately processed by FPGA vendor-specific tools to generate an image that programs the FPGA. Packets received on physical network ports are processed using our logic.
Some examples

• Learning switch
• ICMP echo and TCP ping
• DNS
• Memcachsed
• NAT
Some results

<table>
<thead>
<tr>
<th>Platform</th>
<th>Reference Switch</th>
<th>Emu Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Utilisation</td>
<td>11.42%</td>
<td>12.9 %</td>
</tr>
<tr>
<td>Memory Utilisation</td>
<td>13.23%</td>
<td>13.5%</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>32Gbps</td>
<td>32.7Gbps</td>
</tr>
<tr>
<td>Port-to-port Latency</td>
<td>823ns</td>
<td>825ns</td>
</tr>
</tbody>
</table>
(a) DNS Hit - CDF of Query Latency
(b) Memcached - Throughput
High-level development and debugging of FPGA-based network programs
“Program-hosted Directability” (PhD)

- Program direction

- PhD: transforming programs to host their own directability features.

- “direction feature” becomes a program in constrained language.

- Can invoke/reconfigure these features at runtime.
Original program behaviour

(Normal interaction with external world)

Program

Controller

Director

Hosted directability
trace V max_trace_idx

g();
V = f(X, Y);
N++;
trace V max_trace_idx

g();
V = f(X, Y);
N++;
if V_trace_idx < max_trace_idx then
    V_trace_buf[V_trace_idx] := V;
    inc V_trace_idx;
    continue
else
    inc V_trace_overflow;
    break
Program memory
Location code
Controller’s memory
Original program behaviour

(Normal interaction with external world)

Program

Controller

Director

Hosted directability
if \( V_{\text{trace_idx}} < \text{max}\_\text{trace}\_\text{idx} \) then
\[ V_{\text{trace\_buf}[V_{\text{trace_idx}}]} := V; \]
\[ \text{inc } V_{\text{trace_idx}}; \]
\[ \text{continue} \]
else
\[ \text{inc } V_{\text{trace\_overflow}}; \]
\[ \text{break} \]
<table>
<thead>
<tr>
<th>Command</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>print $X$</td>
<td>Print the value of variable $X$ from the source program.</td>
</tr>
<tr>
<td>break $L \langle B \rangle$</td>
<td>Activate a (conditional) breakpoint at the position of label $L$.</td>
</tr>
<tr>
<td>unbreak $L$</td>
<td>Deactivate a breakpoint.</td>
</tr>
<tr>
<td>backtrace $\langle $ \rangle$</td>
<td>Print the “function call stack”.</td>
</tr>
<tr>
<td>watch $X \langle B \rangle$</td>
<td>Break when $X$ is updated and satisfies a given condition.</td>
</tr>
<tr>
<td>unwatch $X$</td>
<td>Cancel the effect of the “watch” command.</td>
</tr>
<tr>
<td>count ${ \quad$ reads $X \langle B \rangle \langle $ \rangle \quad }$</td>
<td>Count the reads or writes to a variable $X$, or the calls to a function $fname$.</td>
</tr>
<tr>
<td></td>
<td>${ \quad$ writes $X \langle B \rangle \langle $ \rangle \quad }$</td>
</tr>
<tr>
<td></td>
<td>${ \quad$ calls $fname \langle B \rangle \langle $ \rangle \quad }$</td>
</tr>
<tr>
<td></td>
<td>start $X \langle B \rangle \langle $ \rangle$</td>
</tr>
<tr>
<td></td>
<td>stop $X$</td>
</tr>
<tr>
<td>trace ${ \quad$ clear $X$ \quad }$</td>
<td>Trace a variable, subject to a condition being satisfied, and up to trace some length.</td>
</tr>
<tr>
<td></td>
<td>print $X$</td>
</tr>
<tr>
<td></td>
<td>full $X$</td>
</tr>
<tr>
<td></td>
<td>Stop tracing a variable.</td>
</tr>
<tr>
<td></td>
<td>Clear a variable’s trace buffer.</td>
</tr>
<tr>
<td></td>
<td>Print the contents of a variable’s trace buffer.</td>
</tr>
<tr>
<td></td>
<td>Check if a variable’s trace buffer is full.</td>
</tr>
</tbody>
</table>

Table 2. Directing commands making up language $\mathcal{D}$. Note that count has similar subcommands to those of trace, to clear the counters, get their current value, and find out if a maximum value has been reached.
By default, debugging can be done **interactively** using software debugger, which is one of the tools of the trade of any software programmer.

In HDL form, the program can be tested using simulation tools provided for the HDL. By default, debugging involves iteratively rerunning simulations after running tests in **batch**.

Your HDL module can be included in larger infrastructure modules, to yield more accurate simulation results, at the expense of longer simulation times.

By default, debugging hardware is similar to the simulation phase. The output of results is usually further restricted in hardware, due to the absence of a console.
<table>
<thead>
<tr>
<th>Artefact</th>
<th>Utilisation (%)</th>
<th>Performance</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic</td>
<td>Flip-flops</td>
<td>Duration (#cycles)</td>
<td>Latency (μs)</td>
<td>Queries-per-sec (KQPS)</td>
</tr>
<tr>
<td>DNS+ELA</td>
<td>99.74</td>
<td>100.40</td>
<td>57</td>
<td>1.83</td>
<td>1176</td>
</tr>
<tr>
<td>DNS+2e</td>
<td>234.61</td>
<td>151.06</td>
<td>57</td>
<td>1.86</td>
<td>1176</td>
</tr>
<tr>
<td>(Count)</td>
<td>234.46</td>
<td>151.81</td>
<td>62</td>
<td>1.94</td>
<td>1064</td>
</tr>
<tr>
<td>(Trace)</td>
<td>218.30</td>
<td>151.84</td>
<td>70</td>
<td>1.99</td>
<td>1010</td>
</tr>
</tbody>
</table>

Table 4. Utilisation and performance profile of the DNS+ELA against the DNS having one extension point, where the extension point is NOP, packet counting, or variable tracing. Latency is indicated at the 99\(^{th}\) percentile.
<table>
<thead>
<tr>
<th>System</th>
<th>Features</th>
<th>Network/Control</th>
<th>use leftover resource</th>
<th>embed at Source/HDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Sosić 1992) <strong>Dynascope</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>(Goeders and Wilton 2014) <strong>HLS-Scope</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>(Calagar et al. 2014) <strong>Inspect</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>(Panjkov et al. 2015)</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>(Hung and Wilton 2014) <strong>QuickTrace</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>(Koch et al. 1998) <strong>SLE/CADDY</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>(Monson and Hutchings 2015)</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>(Curreri et al. 2011)</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>(Camera et al. 2005) <strong>BORPH</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>(see §2.5) <strong>PhD</strong></td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td>C</td>
<td>H</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1.** Survey of features provided by debugging systems. Blacked-out boxes mean “not applicable”.
Conclusion

• **Goal**: HLS-based development of network programs
Conclusion

- **Goal**: HLS-based development of network programs
- **Currently**: done using HDL or DSL. We use HLS.
Conclusion

• **Goal**: HLS-based development of network programs

• **Currently**: done using HDL or DSL. We use HLS.

• **What’s new**: lifting+shadowing, host support environment, improved debugging support.
Conclusion

• **Goal**: HLS-based development of network programs

• **Currently**: done using HDL or DSL. We use HLS.

• **What’s new**: lifting+shadowing, host support environment, improved debugging support.

• **Relevance**: will help experienced and novice users of FPGAs, at a time when FPGAs becoming more prevalent.
Thank you

naas-project.org
Extra slides
Three examples:

• print
• break
• unbreak
(break $L \langle B \rangle$) $\in \mathcal{C}$ \hspace{1cm} X \in \text{Var}_p
(break $L \langle B \rangle$) $\in \mathcal{C}$ \hspace{1cm} $X \in \text{Var}_p$

\[ p \quad \Box_{\mathcal{C}} \quad p \quad \quad \text{(need differentiating criteria)} \]
\[(\text{break } L \langle B \rangle) \in C \quad X \in \text{Var}_p\]

\[
p \quad \boxed{\in C} \quad p
\]

\[
\begin{align*}
\mathcal{D} &= \{\} \\
\mathcal{C} &= \{\} \\
D_c &= \lambda \mathcal{D}' \quad X \sim N; \\
&\quad \text{print}(N)
\end{align*}
\]
conditional $\langle B \rangle t =$

$$
\begin{align*}
&\begin{cases}
  t & \text{if } \langle B \rangle = \text{true} \\
  \text{if } I_1 == I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
  \text{else continue} & \text{else}
\end{cases}
\end{align*}
$$


\[ \llbracket \text{break } L \langle B \rangle \rrbracket_{SP} = \text{conditional } \langle B \rangle \text{ break} \]

conditional \( \langle B \rangle \ t = \)
\[
\begin{cases} 
  t & \text{if } \langle B \rangle = \text{true} \\
  \text{if } I_1 == I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
  \text{else continue} & \end{cases}
\]
\[
\begin{align*}
\llbracket \text{break } L \langle B \rangle \rrbracket_{SP} &= \text{conditional } \langle B \rangle \text{ break} \\
\llbracket \text{break } L \langle B \rangle \rrbracket &= @L : \{ \llbracket \text{break } L \langle B \rangle \rrbracket_{SP} \} \\
\text{conditional } \langle B \rangle & \quad t = \\
\begin{cases} 
  t & \text{if } \langle B \rangle = \text{true} \\
  \text{if } I_1 == I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
  \text{else continue} & 
\end{cases}
\end{align*}
\]
\[ L \not\in p \quad p \overset{1}{\prec} p' \]

\[ \text{break } L \langle B \rangle \]

\[ p \quad \Box \varepsilon \quad p' \]

\[
\llbracket \text{break } L \langle B \rangle \rrbracket_{SP} = \text{conditional } \langle B \rangle \text{ break}
\]

\[
\llbracket \text{break } L \langle B \rangle \rrbracket = @L : \{ \llbracket \text{break } L \langle B \rangle \rrbracket_{SP} \}
\]

conditional \( \langle B \rangle \) \( t = \)

\[
\begin{cases} 
t & \text{if } \langle B \rangle = \text{true} \\
\text{if } I_1 == I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
\text{else continue} & \end{cases}
\]
\[
L \not\in p \quad p \prec_L p'
\]
\[
\hat{D} = \{ (\text{bp}, L, 1) \}
\]
\[
\begin{array}{l}
\text{break } L \langle B \rangle \\
p 
\begin{array}{c}
\square 
\varepsilon 
\end{array}
\end{array}
\]
\[
[p]_{SP} = \text{conditional } \langle B \rangle \text{ break} \\
[p]_{SP} = \@ L : \{ [\text{break } L \langle B \rangle]_{SP} \}
\]
\[
\text{conditional } \langle B \rangle t =
\begin{cases}
  t & \text{if } \langle B \rangle = \text{true} \\
  \text{if } I_1 == I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
  \text{else continue} & \\
\end{cases}
\]
\[ L \not\in p \quad p \overset{1}{\prec}_L p' \]

\[ \begin{aligned}
\tilde{D} &= \{ (\text{bp}, L, 1) \} \\
\tilde{C} &= \{ SP[L \mapsto \llparenthesis \text{break } L \langle B \rangle \rrparenthesis_{SP}] \} 
\end{aligned} \]

\[ p \xrightarrow{\text{break } L \langle B \rangle} p' \]

\[
\llparenthesis \text{break } L \langle B \rangle \rrparenthesis_{SP} = \text{conditional } \langle B \rangle \text{ break}\\
\llparenthesis \text{break } L \langle B \rangle \rrparenthesis = @L : \{ \llparenthesis \text{break } L \langle B \rangle \rrparenthesis_{SP} \}
\]

\[
\text{conditional } \langle B \rangle \ t = \\
\begin{cases} 
   t & \text{if } \langle B \rangle = \text{true} \\
   \text{if } I_1 == I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
   \text{else continue} & \text{else}
\end{cases}
\]
\[ L \not\in p \quad p \overset{1}{\triangleleft} p' \]

\[
P \overset{\text{break } L \langle B \rangle}{\square} p' \quad \left\{ \begin{array}{l}
\hat{D} = \{ (\langle \text{bp} \rangle, L, 1) \} \\
\hat{C} = \{ \textit{SP}[L \mapsto [\text{break } L \langle B \rangle]_{SP}] \} \\
D_c = \lambda D'. \text{ if } (\langle \text{bp} \rangle, L, 1) \in D' \text{ then } D' \\
\quad \text{else} \\
[\text{break } L \langle B \rangle] \rightsquigarrow \langle L \rangle; \\
(\langle \text{bp} \rangle, L, 0 \mapsto 1) :\in D'
\end{array} \right.
\]

where

\[
[\text{break } L \langle B \rangle]_{SP} = \text{conditional } \langle B \rangle \text{ break } \\
[\text{break } L \langle B \rangle] = @L : \{[\text{break } L \langle B \rangle]_{SP} \}
\]

conditional \langle B \rangle \ t =

\[
\left\{ \begin{array}{ll}
t & \text{if } \langle B \rangle = \text{true} \\
\text{if } I_1 \overset{\text{==}}{=} I_2 \text{ then } t & \text{if } \langle B \rangle = (I_1 = I_2) \\
\text{else continue} & \text{else}
\end{array} \right.
\]
(break $L \langle B \rangle$) $\in \mathcal{C}$

$p \quad \square \mathcal{C} \quad p$
(break $L \langle B \rangle \rangle \in \mathcal{C}$

$\quad \text{unbreak } L$

$p \quad \mathcal{C} \quad p$

$[[\text{unbreak } L]] = @L : \{\text{continue}\}$
(break $L \langle B \rangle \rangle \in \mathcal{C}$

\[
\begin{array}{l}
\begin{aligned}
\mathcal{D} &= \{\}\nonumber \\
\mathcal{C} &= \{\}\nonumber \\
D_c &= \lambda \mathcal{D}' . \text{ if } (\langle \text{bp} \rangle, L, 0) \in \mathcal{D}' \text{ then } \mathcal{D}' \\
&\quad \text{ else } \lfloor \text{unbreak } L \rfloor \rightsquigarrow \lfloor L \rfloor; \\
&\quad (\langle \text{bp} \rangle, L, 1 \mapsto 0) \in \mathcal{D}'
\end{aligned}
\end{array}
\]

where

\[\lfloor \text{unbreak } L \rfloor = @L : \{\text{continue}\}\]